EUROPEAN COMMISSION

SEVENTH FRAMEWORK PROGRAMME

ICT Challenge 6: Mobility, environmental sustainability and energy efficiency
INFORMATION SOCIETY TECHNOLOGIES
Unit G5 - ICT for the Environment

SmartHouse/SmartGrid

Project Acronym

SmartHouse/SmartGrid

Project Full Title

Smart Houses Interacting with Smart Grids to achieve next-generation energy efficiency and sustainability

Proposal/Contract No: EU FP7-ICT-2007-2 STREP 224628

Deliverable D2.1

In-house Architecture and Interface Description

– Update –

Status:	Final
Version:	
Dissemination Level:	PUBLIC
Date:	22.12.2009

Organization Name of the Lead Contractor for this Deliverable: IWES

Status Description

Scheduled completion date[1]:	30.11.2009	Actual completion date[2]:	22.12.2009
Short document description:	This document reports on in-house architecture and interface description of all automatic energy management devices used in the tree field trials.		
Authors:	Koen Kok (ECN), Cor Warmer (ECN), David Nestle (IWES), Gerd Heusel (IWES), Jan Ringelstein (IWES), Patrick Selzam (IWES), Heiko Waldschmidt (IWES), Anke Weidlich (SAP), Stamatis Karnouskos (SAP), Aris Dimeas (ICCS-NTUA), Stefan Drenkard (MVV))		

Contributions	Peer reviews	Partner	Report/deliverable classification:
X		SAP	[✓ Deliverable]
X		IWES	Activity Report
X		MVV	
X		ECN	
X		ICCS-NTUA	
	X	PPC	

Peer review approval :	☐ Approved ☐ Rejected (improve as specified hereunder)	Date:	
Suggested improvements:			

[1] As defined in the DoW
[2] Scheduled date for approval

Table of Contents

List of Figures

Abbreviations

AMI	Advanced Metering Infrastructure
AMR	Advanced Metering Reading
ASCII	American Standard Code for Information Interchange
BEMI	Bi-directional Energy Management Interface
BEMI GW	Bi-directional Energy Management Interface Gateway
BPEL	Business Process Execution Language
CHP	Combined Heat and Power plant
CRM	Customer Relationship Management
DPWS	Devices Profile for Web Services
DSO	Distribution System Operator
ERP	Enterprise Resource Planning
HTML	HyperText Markup Language
IEC	International Electrotechnical Commission
IP	Internet Protocol
IPv6	Internet Protocol Version 6
IT	Information Technologies
KNX	Konnex(-standard)
MMS	Manufacturing Messaging Specification
OASIS	Organization for the Advancement of Structured Information Standards
OSGi	Open Services Gateway initiative
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
RAM	Random-Access Memory
RFID	Radio Frequency Identification
SCM	Supply Chain Management
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	State of Charge
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UDP	User Datagram Protocol
WS	Web Service
WS-DD	Web Services Discovery and Web Services Devices Profile

1. Introduction and Overview

1.1. In-House Architecture Overview

Deliverable D1.2 “Technology Trends” already provide an overview of the in-house technology required. In general, the in-house architecture consists of:

- One or several communication/home automation systems such as KNX, ZigBee, Z-Wave, etc.
- Intelligent nodes/agents that perform communication and control operations over these communication systems. In some cases, these nodes just perform basic control functions such as temperature surveillance or switching commands from the home automation systems, in other cases (such as the PowerMatcher concept), each node is a real intelligent agent.
- In most cases, a dedicated communication gateway to the outside world exists. In concepts such as BEMI, the communication gateway is at the same time the in-house manager, in other concepts it is just a communication gateway without higher control capabilities than other agents in the house.
- Several devices operated by the customer and measurement nodes. In general, the meter can be considered a measurement node though having a special role for most business cases.
- User Interface

It is a goal of the SmartHouse/SmartGrid system to enable automatic identification of home appliances. To make this possible, several services for registration, management and access of devices and other hardware/software resources available in the house have to be defined (see Chapter 2). Also, standardized data representations of household appliances relevant to smart grids have to be developed (see Chapter 3). For each communication system it must be determined individually how these data models can be used and implemented. It is described how these models are transmitted and used by Web Services (for home automation and internal services of embedded systems, Web Services might – as of today – not be well suited; for this reason, the field implementations use various subsets of the services and data models described here, based on technologies such as Java, OSGi[3] and the .Net framework).

1.2. In-House Architecture Design Approach

The main goals for the design of the in-house architecture are:

- to provide an environment for applications in the area of energy management and energy efficiency at customers‘ sites in smart distribution grids,
- to allow for access to devices and other hardware functionalities that are connected to the system via standardized data models or device service models,
- to allow for automated registration of new devices based on standardized data models and device services,
- to make the data provided from outside the BEMI that might be relevant to various applications (such as the price of electricity) accessible based on standardized data models,
- to define standardized services of the framework for using these data models and device services, and
- to also provide standardized services for functionality that will be needed for many applications: the user web interface, persistent storage of certain types of data and logging.

From these goals, several architectural elements of the framework have been identified and defined (see also Figure 1):

[3] OSGi Alliance http://www.osgi.org/

Application

An Application is a piece of software that is able to run in the environment of the in-house framework. In contrast to a communication system driver, it is not used to enable the physical connection to hardware. Applications represent certain use cases and should be specific to a use case.

Resource

A Resource is a representation of states, parameters or other data generated outside the system. So a Resource can either represent a physical device, a communication system and its parameters/state or data transmitted to the system from a control station, such as a price profile.

Resource Type

A Resource Type is a model definition for resources. In order to enable automated device identification and plug&play, standardized resource types have to be used on all framework implementations. However, it shall be possible to add new Resource Types to a framework when standardized types are available. In an object oriented perspective, this is the class description of which the resources are instances.

Communication System

A Communication System is able to connect the data representation of a Resource with the actual physical device it represents or with the external data source (e.g. the control station delivering the price profile). In this way, the information of the physical connection of each resource is made transparent to the rest of the framework as it is processed solely by the Communication System. Each connection links one data element of a resource to an address of a communication system. The addressing scheme of each Communication System is specific to each Communication System, of course.

APIService

The framework needs to offer several functionalities to the Applications and Communication Systems. These services can be grouped into the administration of Resources (Resource Administration), the administration of Applications, the system time they are using and the way they are executed (Time Control), services for persistent storage of preferences data of Applications and of data structures that are commonly needed by applications in the area of energy management and efficiency (Persistent Storage), access to a user interface and services for logging and evaluation of text log messages as well as of measurement data series. The APIService is the entity of all modules of services of the framework. Further services available to Applications and Communication Systems can be provided by Applications, but the services of the APIService can be expected on every framework implementation, thus being a base set for interoperability.

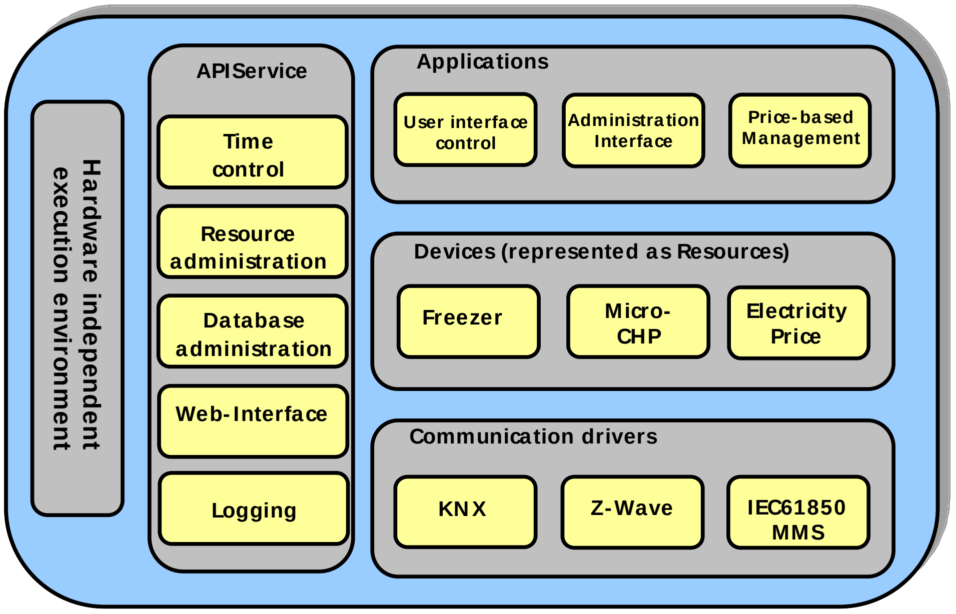


Figure 1: In-house framework overview

1.3. Concept and Standards of Web Services

1.3.1. Overview of Integration

The SmartHouse will need to interact with numerous external entities, let it be alternative energy resources, marketplaces, enterprises, energy providers etc. The de-facto standard for high-level communication today is via web services, which allows for flexible functionality integration without revealing details for the implementation. Therefore, the heterogeneity is hidden, while a common service-based interaction is empowering the creation of sophisticated applications.

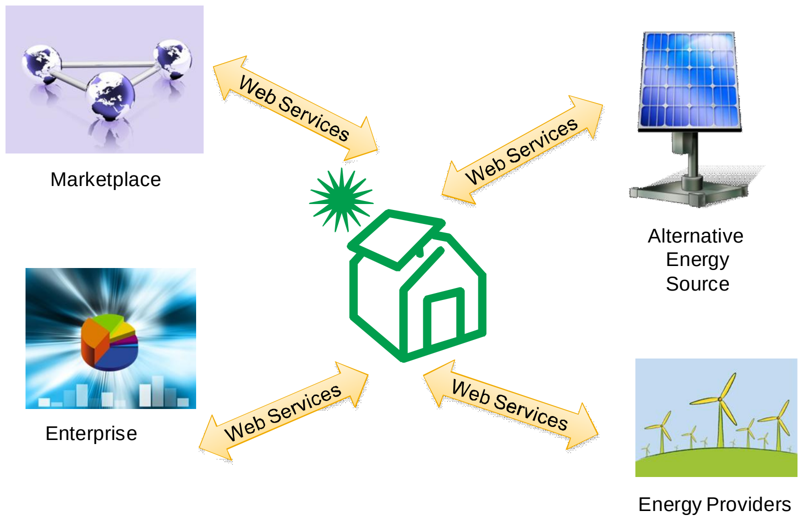


Figure 2: Service interaction of the SmartHouse with external entities

The SmartHouse/SmartGrid project is deeply investigating the possibility of using web services at least for the interaction of the SmartHouse with other SmartHouses, and with entities in a SmartGrid.

1.3.2. *In-House Device Interaction*

We already mentioned that web services are the de-facto standard for high-level and cross-enterprise cooperation. Within the SmartHouse, we have numerous protocols and even different technologies at the hardware communication layer. It is, however, a common belief that all of this heterogeneity will be hidden behind gateways and mediators, which will eventually allow the device to tap into an IP-based infrastructure, using Internet standards. Already today, the IP protocol is developed further to run in tiny and resource constrained devices (6lowpan), while with the IPv6 (and 6lowpan), any device will have its own IP address and be directly addressable.

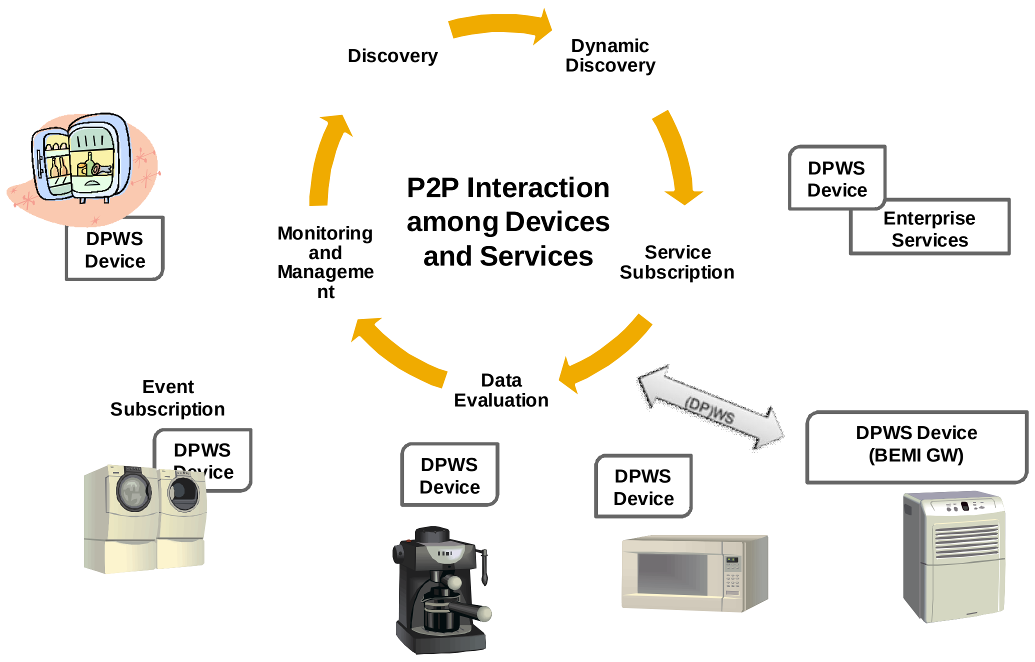


Figure 3: Device Peer-To-Peer interaction in the SmartHouse

Due to IP penetration down to discrete device level, it is expected that devices will not only provide their information for monitoring to controlling entities, but will be able to dynamically discover nearby devices and collaborate with them (as depicted in Figure 3). In this way, P2P interactions will emerge, which can be exploited by locally running applications that execute monitoring or controlling tasks.

Devices in the SmartHouse are and will remain highly heterogeneous, both in hardware and in software. As such, we need to find a way that this heterogeneity is abstracted, and yet communication (and collaboration) among them can be achieved. The development of middleware approaches that act as the “glue” for device to business connectivity (and later also for device to device connectivity) is a viable approach. The SmartHouse/SmartGrid project intends to use such an approach, as depicted in Figure 4.

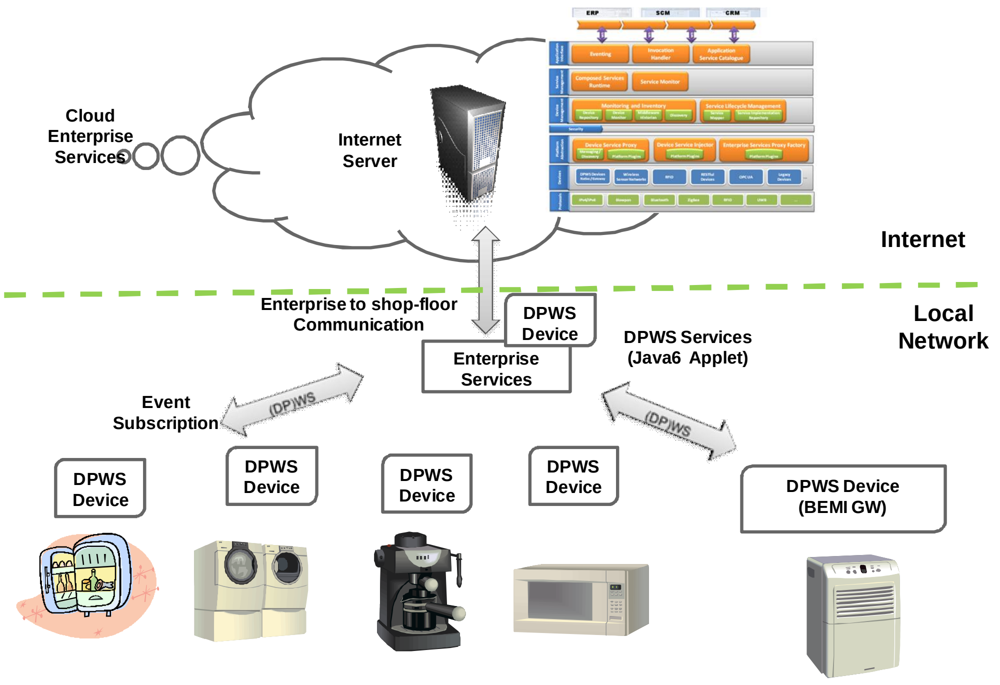


Figure 4: Enterprise assisted device interaction in the SmartHouse (see Figure 6 for details)

In parallel to local collaboration, devices with advanced capabilities will be able to interact with network-based services hosted in enterprise systems, or simply somewhere on the Internet. These devices will be able to simply enhance their own functionality in a dynamic way by simply invoking services that were not thought of at the time of device design. As an example, price signals are often brought up as a key functionality. In this example, the device would get a price signal from the energy provider and adapt its operation accordingly.

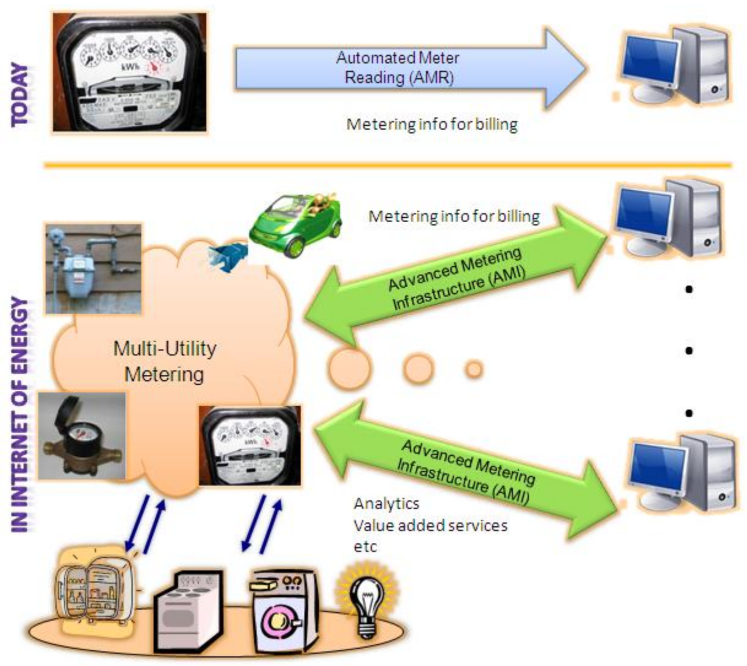

TODAY
Automated Meter Reading (AMR)
Metering info for billing
IN INTERNET OF ENERGY
Metering info for billing
Advanced Metering Infrastructure (AMI)
Multi-Utility Metering
Advanced Metering Infrastructure (AMI)
Analytics
Value added services
etc

Figure 5: Smart meter and device collaboration beyond billing

1.3.3. *Enterprise Integration Architecture*

The Enterprise Integration Architecture (as depicted in Figure 6) enables enterprise-level applications to interact with and consume data from a wide range of networked devices using a high-level, abstract interfaces that features (web) services standards. Those standards already constitute the common communication method used by the components of enterprise-level applications. Web services are the technology canonically used to implement business processes, which are frequently modelled as orchestrations of available web services. This allows the connected, networked devices to directly participate in business processes while neither requiring the process modeller nor the process execution engine to know about the details of the underlying hardware.

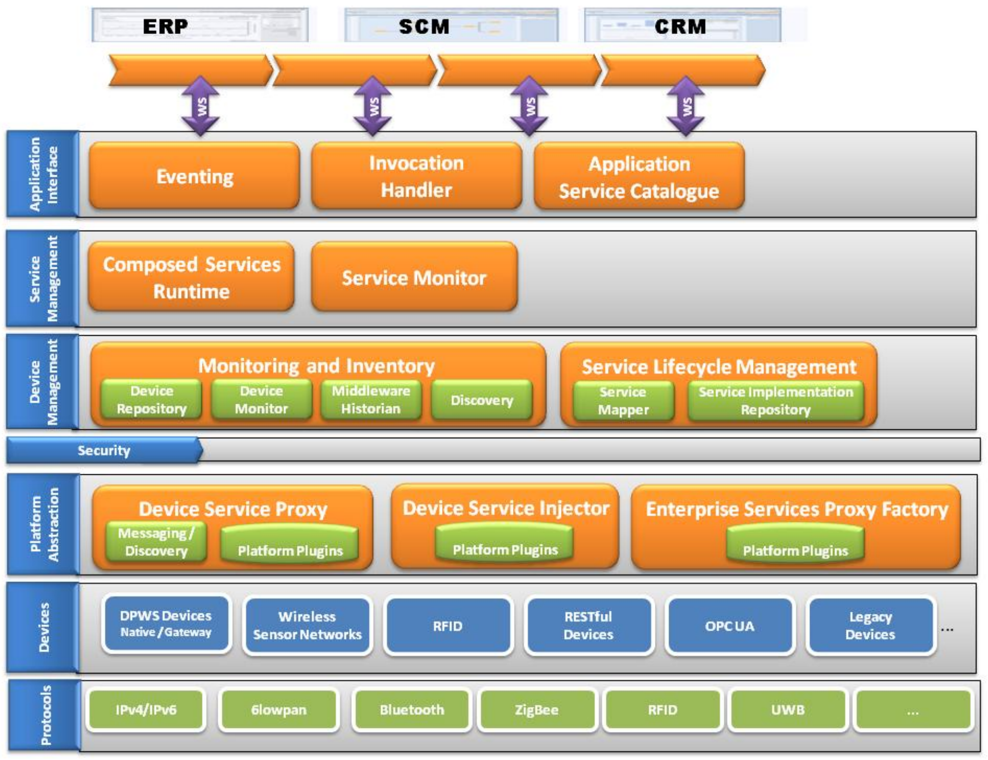


Figure 6: Enterprise Integration Architecture

The architecture implemented hides the heterogeneity of hardware, software, data formats and communication protocols that is present in today's embedded systems. The following layers can be distinguished: application interface, service management, device management, security, platform abstraction, protocols and devices.

- Application Interface: This part of the integration architecture features a messaging (or eventing) system, allowing an application to consume any events whenever it is ready to and not when a low-level service happens to send them. A so called invoker allows buffer invocations to devices that are only intermittently connected. Finally, a service catalogue enables human users and applications to find service descriptions and pointers to running service instances. Both atomic services hosted by the devices and higher-level composed services are listed here.
- Service Management: All functionality offered by networked devices is abstracted by services. Either devices offer services directly or their functionality is wrapped by a service representation. On this layer of the integration architecture and on all layers above, the notion of devices is abstracted from, and the only visible assets are services. An important insight into the service landscape is to have a repository of all currently connected service instances. This is provided by the service monitor.

This layer also provides a runtime for the execution of composed services. We support the composition of business processes primarily by offering an execution service for underspecified BPEL processes, meaning that service compositions can be modelled as business processes where the involved partners do not need to be explicitly specified at design time.

- Device Management: All devices are dynamically discovered, monitored and their status is available to the enterprise services. Furthermore, it is possible to remotely deploy new services during run-time, in order to satisfy application needs.
- Security: Both devices and back-end services may only be accessed by clients that have a certain role and provide correct credentials that authenticate themselves. This layer implements the correct handling of security towards the devices and the enterprise-level applications.
- Platform Abstraction: As stated before, devices either offer services directly or their functionality is wrapped into a service representation. This wrapping is actually carried out on the platform abstraction layer. In the best case, a device offers discoverable web services on an IP network. In this case, no wrapping is needed because services are already available. If the device type, however, does not have the notion of a service (it might use a message-based or data-centric communication mechanism), the abstraction into services that offer operations and emit events can be a complex task. In addition to services enabling the communication with devices, this layer also provides a unified view on remotely installing or updating the software that runs on devices and enables the devices to communicate natively, i.e. in their own protocol with back-end devices.
- Protocols: We expect that heterogeneity in communication protocols will exist also in the future as it serves different domains and respectively devices.
- Devices: Heterogeneous devices are expected to connect to the architecture. These include industrial devices, home devices, or IT systems such as mobile phones, PDAs, production machines, robots, building automation systems, cars, sensors and actuators, RFID readers, barcode scanners, or power meters. We used several of the listed types of devices during prototype implementations.

1.3.4. Gateway and Service Mediator

Gateways and mediators are used to integrate low capability or legacy devices. Lately with the emergence of SOA we are also witnessing gateways and specialized mediators ("Service Mediators") that offer the functionality of the devices they abstract as a service.

- The Gateway: A Gateway is a device that controls a set of lower-level non-service-enabled devices, each of which is exposed by the Gateway as a service-enabled device. This approach allows to gradually replace limited-resource devices or legacy devices by natively WS-enabled devices without impacting the applications using these devices. This is possible since the same (web service) interface is offered this time by the (service-enabled) device and not by the Gateway. This approach is used when each of the controlled devices needs to be known and addressed individually by higher-level services or applications.
- The Service Mediator: Originally meant to aggregate various data sources (e.g. databases, log files, etc.), the Mediator components evolved and are now used to not only aggregate various services, but possibly also to compute or process the data they acquire before exposing it as a service. Service Mediators aggregate, manage and eventually represent services based on some semantics (e.g. using ontologies).

  Service Mediators aggregate various heterogeneous devices so that higher level application could communicate to Service Mediators offering WS, instead of communicating to devices with proprietary interfaces. The benefits are clear, as we don't have the hassle of (proprietary) driver integration. Furthermore now processing of data can be done at Service Mediator level and more complex behaviour can be created, that was not possible before from the standalone devices.

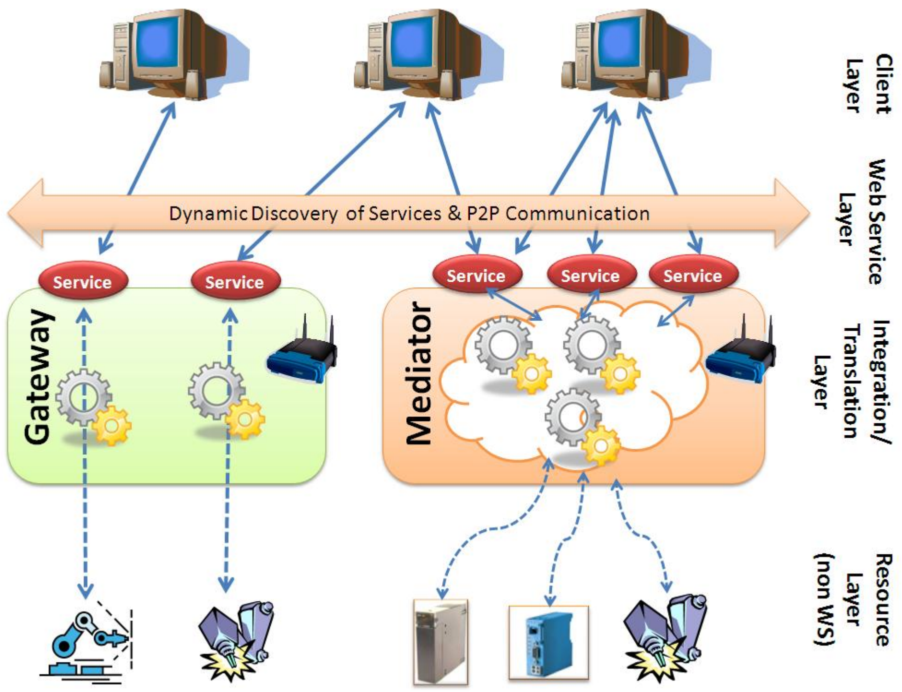
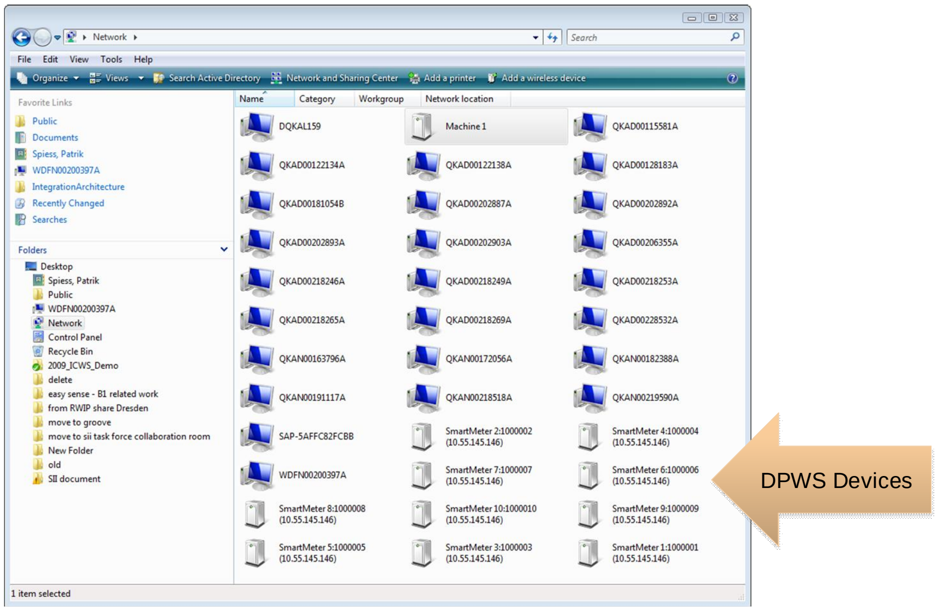


Figure 7: The gateway and (service) mediator concepts

1.3.5. Device Profile for Web Services (DPWS)

DPWS defines a minimal set of implementation constraints to enable secure web service messaging, discovery, description, and eventing on resource-constrained devices. DPWS is an effort to bring web services on the embedded world taking into consideration its constrained resources. Several implementations of it exist in Java and C, while Microsoft has also included a DPWS implementation (WSDAPI) by default in Windows Vista and Windows Embedded CE.



DQKAL159	Machine 1	QKAD00115581A
QKAD00122134A	QKAD00122138A	QKAD00128183A
QKAD00181054B	QKAD00202887A	QKAD00202892A
QKAD00202893A	QKAD00202903A	QKAD00206355A
QKAD00218246A	QKAD00218249A	QKAD00218253A
QKAD00218265A	QKAD00218269A	QKAD00228532A
QKAN00163796A	QKAN00172056A	QKAN00182388A
QKAN00191117A	QKAN00218518A	QKAN00219590A
SAP-5AFFC82FCBB	SmartMeter 2:1000002 (10.55.145.146)	SmartMeter 4:1000004 (10.55.145.146)
WDFN00200397A	SmartMeter 7:1000007 (10.55.145.146)	SmartMeter 6:1000006 (10.55.145.146)
SmartMeter 8:1000008 (10.55.145.146)	SmartMeter 10:1000010 (10.55.145.146)	SmartMeter 9:1000009 (10.55.145.146)
SmartMeter 5:1000005 (10.55.145.146)	SmartMeter 3:1000003 (10.55.145.146)	SmartMeter 1:1000001 (10.55.145.146)

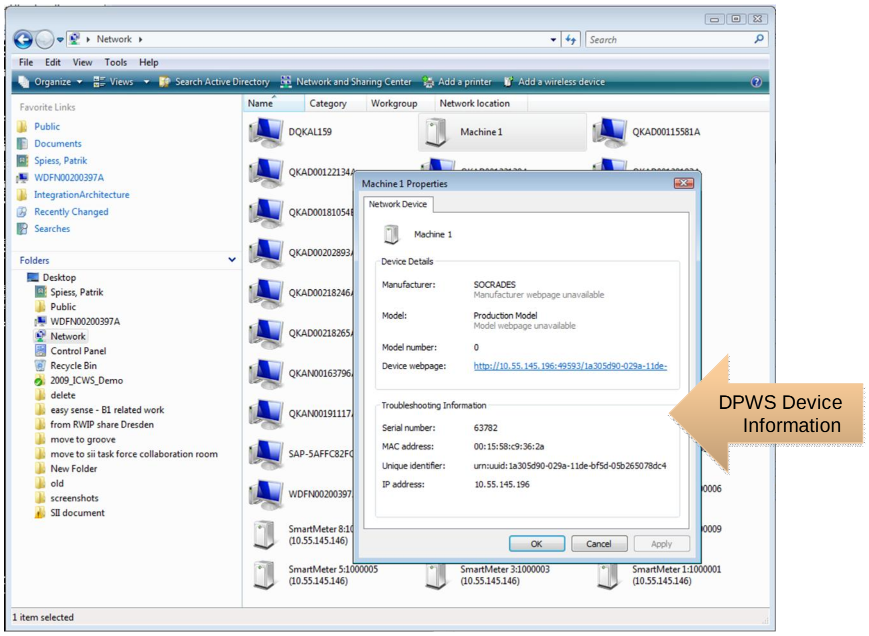
1 item selected

Figure 8: Dynamic discovery of DPWS devices in Windows Vista

In August 2008, the OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) Technical Committee was created to further advance the existing work, e.g. DPWS. On May 13, 2009, the OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) Technical Committee unanimously approved the following specifications as Committee Specifications:

- Web Services Dynamic Discovery (WS-Discovery) Version 1.1
- Devices Profile for Web Services Version 1.1
- SOAP-over-UDP Version 1.1

The specifications have now been submitted for consideration as OASIS standards. It is expected that the announcement of the submission will be made to the full OASIS membership on June 1, 2009, and the voting period will be June 15-30, 2009.


Network
File Edit View Tools Help
Organize Views Search Active Directory Network and Sharing Center Add a printer Add a wireless device
Machine 1 Properties
Network Device
OK Cancel Apply
DPWS Device Information

Machine 1

Device Details

Manufacturer:	SOCRADES Manufacturer webpage unavailable
Model:	Production Model Model webpage unavailable
Model number:	0
Device webpage:	http://10.55.145.196:49593/1a305d90-029a-11de-

Troubleshooting Information

Serial number:	63782
MAC address:	00:15:58:c9:36:2a
Unique identifier:	urn:uuid:1a305d90-029a-11de-bf5d-05b265078dc4
IP address:	10.55.145.196

Figure 9: Asset management and direct device information via DPWS

2. In-house service definitions

2.1. Modelling of resource types

To make the platform flexible for new applications and new devices it must be possible to add new device types to the system during runtime. The description of these new device types shall use standard resource types wherever possible. If, for example, a device type “wine cabinet” shall be added to the system, the new resource type “Wine Cabinet” shall use the existing types “Electrical Switch” and “Temperature Storage” as part of the model. In this way, an application performing management of dives containing a thermal storage that does not know the type “Wine Cabinet”, but knows the types for the switch and the temperature storage, can still control the new device type just by addressing the known sub-resources.

To allow for dynamic adding of resource types a modelling language has to be defined. If the framework is implemented using an object oriented language, the easiest way for this language is to describe the resource type as a class of this language.

In order to allow for a general processing of new resource types, the type description should not contain any services, but all services that can be performed on resources should be defined by the resource administration. So each resource type can only contain data elements that are either of “basic type” or of another resource type that has been defined before. For the scope of this in-house architecture five basic resource types are considered sufficient:

- Boolean
- Integer
- Analogue (most Analogue values need a physical unit. Each implementation must specify how the units are defined)
- String (character chain either coded as ASCII or Unicode)
- Time (may represent a point in time or a time duration)

The type “Enumerated” is used for variables for which several options are specified. How these options are represented in a language-specific specification is not defined here.

An array in this context is a list of data entities of the same type that are accessed via an index.

Additionally the framework has to define references to various data types required by the services of the framework. These references might by names, id’s or references to objects depending on the implementation. A list of these reference types is given in Annex II.

For each basic element of a resource type it must be specified whether the value is a parameter or any other value that needs to be stored in the persistent storage or whether the value is a measurement value that is updated continuously and thus should not be written into the permanent storage every time it is changed.

The first data model to be defined is the data structure defining a resource type description:

Resource Type Description	TypeDesc		OSGI-impl.	
Name/Description	Type	Persistent	Name	Unit
name	String	yes	name	-
number of elements (defining length of Arrays)	Integer	yes	-	-
Data elements	Array of DataElement	yes	dataElement	-

with the structure DataElement:

Element of a Resource Type Description	DataElement		OSGI-impl.	
Name/Description	Type	Persistent	Name	Unit
name	String	yes	name	-
type	ResTypeRef	yes	type	-
determine whether element is persistent	Boolean	yes	persistent	-

Table 1: Resource Type description

Further data models (device models, other management data models such as pricing and metering data, system data models) are given in the Annex.

2.2. Services of the Resource Administration of the Framework

Access and administration of resources, resource types and communication systems are performed by the Resource Administration. The most important services of the Resource Administration comprise (see Annex for details):

- Management of resource types
  - add/delete resource type
  - get all resource types registered, get elements of a resource type
- Management of resources
  - add/delete resource: When a resource is added it must be based on a resource type that was registered before.
  - get resource name/type
  - get super/top-level resource: Most devices contain sub structures represented by sub resources. This service allows to step up in a tree of resource elements or just get the top-level resource representing an entire device
  - get a certain child resource based on its name
- Search for resources
  - search for all resources fitting a resource type description
  - search for all resources that are sub resources of a resource
- Read / write of resources
  - read from element of a resource (in general this works only for data elements that are stored persistently; volatile measurement values should be acquired by registering an appropriate resource demand for reading.)
  - write into element of a resource
- Management of communication systems
  - add/delete communication systems
  - get communication systems installed
- Management of the resource demand of applications
  - register resource demand: applications usually control resources of one or several defined resource types (e.g. a freezer energy monitoring application) or devices that contain certain sub-resources (e.g. a temperature storage for a state-of-charge (SOC) based energy management). Using this service an application can tell the framework that they are able to read from such devices. In this case the application is informed as soon as such a device is available and also when the device is disconnected again. In case the application also needs to control the device (write to the device), a decision has to be made which application gets control on the device in case several applications are registering the same type of demand. This decision is made by the resource administration based on the priority of each resource demand.

    It must be specified by each language implementation how the application is informed of new resources fitting a demand, changes of the demand fulfilment and changes of the values of the resources that are assigned to an application (e.g. event/messaging or call-back specification).
  - register secondary demand: in some cases an application only needs access to a device when another resource demand can be fulfilled. It may even be the case that a resource must fit to two different resource demands at the same time. For example the management of SOC-based devices is only possible if they contain a temperature storage and at the same time a switch that can be controlled by the management. This service allows adding such a secondary demand to another demand registered before.

In many cases a resource type can only describe the most basic attributes of a device type. To make this resource description extensible, additional sub-resource types may be defined that can be added to a resource of a certain resource type. These resource types are called “Additional-Types”. For example the general freezer model may just contain a switch and a temperature storage. An additional type of the resource type “Freezer” may be characterized by a resource type “Door Closed Sensor”. Additional services are required to allow for the handling of additional-types:

- add/delete additional-type
- add sub-resource of additional-type to a existing resource

2.3. Services of the Persistent Storage

The main function of the persistent storage is to store data that needs to be recovered after a shutdown and restart of the framework and the applications. Typically this is the function of a database. In order to make access usage as simple as possible and to meet the specific needs of energy management and efficiency related applications, the framework offers specialized data structures. Three main types of data are considered relevant for the scope of this framework:

- “Preferences” (=”properties”/”parameters”/”settings”):
  Set of values and arrays of basic values that can be accessed by two names (Preferences Set name and Preference name). As the name indicates this is mainly intended to store preferences/properties/ parameters of the applications. When a property is read, a default value should be provided by the application, which should be returned in case the property has not been set before.
- “Schedule Series”:
  Time-based series of data of that is provided in uniform time ranges such as minutes, days or years. Currently, only fixed-step-schedules are considered meaning that each type of schedule defined by a “Schedule Series Definition” object has equal time steps of a certain time span. Schedules are used e.g. to represent price profiles, metering profiles and device schedules. Each language-specific specification of the framework must specify to which time zone the schedules should be aligned. Usually, this should be the local time zone of the system so that most applications do not have to worry how they are situated relative to the schedule alignment (except for applications that need global time).
- “Time Series Set”:
  Each Time Series Set contains one series of time stamps. Furthermore, it contains time series of data, usually measurement values. For each time stamp, exactly one value exists for each time series of data. This allows for efficient storage of measurement data that shall be logged with a certain frequency. A time series can be opened and closed for writing. However, the number and name of the data series can change when a time series set is re-opened. In this case, a time series may end at some point or start. This is necessary as the exact configuration of sensors that should be logged based on a certain frequency may change (see description of services for details).

The following data structures are required for operating on these types:

Time Series Set definition	TimeSeriesSetDef
Name/Description	Type
Number of time series in the Time Series Set (defining length of Arrays)	Integer
Names of time series	Array of String
Start time of Time Series Set	Time

Time Series Set definition	TimeSeriesSetDef
Name/Description	Type
Reduction mode (the framework should support the reduction of the data stored compared to the data written into the Time Series Set): • no reduction • under-sampling (has the same effect as writing to the time series with a lower frequency) • mean • maximum deviation from standard value	Enumerated
Reduction factor	Integer
Duration data may be stored in non-permanent storage before written to permanent storage. This may be used in order to optimize performance by reducing writing operations to a hard or solid state disk.	Time
Maximum number of time stamps within Time Series Set. If more time steps are collected, the number of elements shall be cleaned up as defined below	Integer
Clean-up/delete mode: • delete oldest data ("ring buffer") • make sure that 50% of the maximum number of time stamps is maintained with all data provided, the rest should be thinned out so that the time range to the oldest time stamp is at least 10x the time range to the 50% time stamp • (may be extended)	Enumerated

Schedule Series Definition	ScheduleDefinition
Name/Description	Type
Type: May be Boolean, Integer or Analogue. Time- and String-schedules are not considered relevant to the scope of this framework.	BaseType
ScheduleTiming: decides whether the time stamp for each value is determined by: • the start time and a fixed step size • absolute time stamps • time stamps relative to the start time of the schedule	Enumerated

Schedule Series Definition	ScheduleDefinition
Name/Description	Type
Interval covered by one schedule, options must be at least • Year : this means that even fixed-step-schedules vary in absolute length due to leap years • Year with 366 days, final values may be meaningless • ¼-year this means that even fixed-step-schedules vary in absolute length • ¼-year with 92 days, final values may be meaningless • Month: this means that even fixed-step-schedules vary absolute length • Month with 31 days, final values may be meaningless • Week • Day • Hour • Minute	Enumerated
Switch with daylight savings time: • schedule alignment switches with beginning and end of daylight savings time • schedule does not switch with daylight savings time. If schedule are aligned to a local time switching to daylight savings time, the schedule shall always be aligned to winter time.	Enumerated
Value returned of schedule is not available for a certain time step	Must be able to represent Boolean, Integer and Analogue
Step size (for schedules using a fixed step size only)	Time

Elements of a single schedule with fixed time step (other schedules are not specified yet), which contains the data for one interval specified for a single schedule in the corresponding Schedule Series definition:

Fixed Step Schedule (single schedule)	ScheduleFixedStep
Name/Description	Type
corresponding Schedule Series definition	ScheduleDefinition
start time of schedule	Time
values of schedule	Array of <Type according to sche-dule definition>
backup type: if true, the schedule is a backup schedule	Boolean
Backup condition (only relevant for backup schedules): see dissertation thesis of David Nestle for details	Integer

The following services are required to operate on these three types of persistent data (see Annex for details):

- Preferences
  - create/delete parameter set
  - add/delete parameter to parameter set
  - read/write parameter
- Schedule Series
  - create/delete schedule series definition
  - get schedule series definition
  - add schedule: new scheduled information is added
  - get schedule: The schedule shall be built based on regular schedules added for the scheduling period and backup schedules if necessary
  - get value from schedule: Based on a time specified the corresponding scheduled value is returned
  - get type of schedule (Boolean, Integer, Analogue)
- Time Series Set
  - create/delete Time Series Set Definition
  - open/close Time Series Set: When a Time Series Set is re-opened the time series that are included shall be specified. In case a time series is already known in the Time Series Set, it shall be continued. Otherwise it shall be added. In case a time series previously declared, but not included in a re-opening, new values cannot be added until the Time Series Set is closed and re-opened again. The time series shall be kept in the Time Series Set for further re-openings, however.
    This means that a Time Series Set contains exactly one value per time step for each time series included in the interval of one opening. Time series not included in a certain opening do not have values for this interval, however.
  - get Time Series Set definition of a Time Series Set
  - add time stamp and one value for each time series opened into a Time Series Set
  - change value already written into Time Series Set
  - read values of one or several time series within an interval of time stamps specified from one Time Series Set (this should be possible independently of whether a Time Series Set is opened or not)
  - read values of one or several time series from one or several Time Series Sets within an interval of time stamps specified. Usually the different Time Series Sets do not have the same time stamps within the interval, so an alignment of the time stamps has to be specified.

2.4. Logging Services

Logging services are grouped into two sub-functionalities:

- Logging of text messages
- Data series logging

The first area is relatively simple and just requires a method to send log messages with fields for the sender and the logging/service level into a log file, which should also contain the time when the log message was generated.

For the area of data logging the following services are required (as these services are still under development they are not specified in more detail in Annex yet):

- register data element from resource for logging (Input: data element, frequency of logging value (should support a default frequency and a trigger signal from the measurement system indicating that a new value is available), parameters of Time Series Set regarding reduction, storage cleanup, RAM-buffer usage etc.) All logged values with the same logging frequency should be stored in the same Time Series Set using the parameters specified.
- unregister data element from logging

- get logging configuration for a certain data element
- get access to logged data for a resource (return Time Series Set and time series)
- get all data elements that are currently logged
- generate evaluation from logged data of a data element (at least mean, standard deviation, minimum and maximum should be supported)

Furthermore, it should also be possible to calculate certain statistical data without storing entire data logs. In this case for each new measurement value a statistical value is updated by an appropriate algorithm, but the new value itself does not have to be stored.

- register continuous evaluation for a data element
- unregister continuous evaluation for a data element
- get evaluation configuration for a data element
- get access to evaluation results for a data element
- get all data elements that are currently evaluated

The framework should also support processing of events that are generated by applications for logging purposes. Usually events should be counted and written into the text message log.

- register event type
- unregister event type
- report event to the logging service: The event type must be known to the system, events may be connected to a resource.
- get number of events counted of a certain type
- evaluation of number of events based on a type/resource specification

2.5. Services of the User Web Interface

The user web interface consists of the administration interface (accessible only for the administrator), web pages installed by applications and a common start page. Usually, applications should not control the entire browser pane, but a fixed navigation area giving access to all applications and showing “Favourites” should be included. The web interface preferences should contain the following support and pieces of information (as these services are still under development, they are not specified in more detail in Annex, yet):

- Can the user web interface accessed via the internet?
- Should the user web interface always start from the common start page, from another page or from the page last viewed?
- Control of favourite navigation bar (may be set by user, most frequently used applications, determined by frequency of usage of single page or by an application)
- Port on which user web interface can be accessed (IP configurations should be made via an appropriate communication system)
- Configuration of SSL/TLS if applicable

The preferences may be editable via the administration interface. The common start page and the application pages are usually accessible without login from the local network and via a list of defined logins from the internet.

Services of the web interface for applications are:

- Register/delete web resource (HTML with dynamic content, picture, file for download, …)

The programming environment to be used to express dynamic web page content shall be specified by the framework.

2.6. Runtime Control of Applications and Communications Systems

The framework must support operation of several applications and communication systems in parallel. The runtime control of these applications will usually somehow be controlled by the framework. The module of the framework performing this task is called the “Time Control”. There are two fundamental concepts available in most operating systems to implement this: Each application could run in a separate thread or all applications could run in a common thread with a step function of each application being called from the time control.

In many cases, synchronous operation of applications and communication systems has advantages. Usually, applications need input from communication systems and vice versa. So it makes sense to perform exactly one iteration of an application after one iteration of the communication systems. This is ensured with the second approach explained above. If the application directly reacts on the call-back method, events or interruptions that are invoked from the resource administration as soon as a new measurement value is available can also be solved using separate threads, though.

The second approach also avoids almost all problems regarding timing of application interaction as every application can assume that it will be able to finish the current iteration before any other application accesses any common data or starts any call-back function. When all write operations on resources require that the resource demand has been registered and approved before writing, this is not any issue anyways, however.

The second approach also makes debugging of a set of applications easier as the order of execution is predictable. For this reason, it is recommended that a framework supports step functions being called from the time control.

There are also some problems with this approach, though. In case an application has idle time during its step function, this approach will lead to idle time of the entire framework as other applications can only start their operation after the step function of this application is finished. For this reason a combination of both approaches will be necessary.

This can be done in two ways:

- Applications with idle time start a separate thread for the relevant part of the program. The separate thread shall not perform any asynchronous write, but hands the result of each write operation over to the step function of the application, which performs the write operation of the latest values available synchronously. From the perspective of the time control, this does not require any additional functionality. If reading by the application needs to be synchronized this usually is the only way of implementation as the step function can perform the relevant reading and store the values for the asynchronous part of the application. In case the registration of resource demand mechanism ensures that only one application is allowed to write to a certain resource at a time, also writing from a separate thread is no problem.
- Applications can tell the time control that they need to operate asynchronously. The time control calls the step function in a separate thread. Write operations from these applications are stored, but not executed by the Resource Administration before the end of a synchronous cycle of the time control. This requires additional functionality of the framework. The advantage is the fact that programming the application does not require additional effort. In case the registration of resource demand mechanism ensures that only one application is allowed to write to a certain resource at a time there is no need for this option.

The second problem occurs in case a malfunctioning application does not return from a step function. In this case, all synchronous applications are blocked and do not operate anymore. For this reason, safety-relevant applications should start their own thread. It is also recommended to implement some kind of watch dog functionality, which restarts the system in case a step function is blocking for a certain time. The blocking application should not be started anymore after the restart of the system and the administrator should be informed automatically. In this way malfunctioning applications can be detected quickly, which can be considered an advantage of the approach.

2.7. Services of a communication system

Each communication system must implement a number of services that are used by the resource administration and the administration interface. These services are (see Annex for details):

- get name
- get minimal time step between two cycles of acquiring data
- check if minimal time step varies (e.g. based on the number of sensors that are connected via the communication system)
- set/get standard time step for new sensors (the standard time step must be equal or greater than the minimal time step allowed, of course)
- get configuration data / notification on update of configuration data
- check whether an address is valid for the communication system
- connect a resource data element to an address, should return a ComConnectionRef (used if the communication system does not support plug&play itself so that device resource representation has to be registered and connected by the user itself or by an application)
- close connection between data element and an address
- request to update a connected data element based on the ComConnectionRef

In case the communication system is acting as a server (waiting for connections from the outside, not able to start communication requests actively), the time step between two cycles of acquiring data and a request for update of a connection usually has no meaning unless an outside client supports such a parameter and allows for server controlled polling/pushing. When the communication system represents a server; an address shall represent the address on which clients can target a resource.

Usually; each data element should not be connected to more than one communication system. In case a data element is connected to two communication systems, the framework acts as a gateway.

2.8. Security

The framework must include an administrative interface. This interface must support a role-based access model. If access to the administrative interface is implemented using a public network, all data transmission must be encrypted. SSL/TLS is suggested as a standard for a web based user interface. If the web interface can only be accessed from the local network, encryption is not considered necessary.

The framework should also support different roles, at least “Customer”, “Energy Supplier” and “Distribution System Operator (DSO)”. Further possible roles could be service support for various devices, metering company etc. Applications should be signed by the owner of a role thus connection the application to a role.

In the user interface; it shall be possible to assign permissions to read or to read/write a device to a role. The framework should check if an application has the right to access a resource in the desired read/write-mode based on the permissions of the role that signed the application before installation.

It is not assumed possible to prevent a malicious application from causing harm when installed in the framework. The prevention from malicious applications shall be performed by checking the signature from a trusted owner of a role. The permissions and limitations assigned to the owner of a role means that applications provided by the owner cannot access resources against the will of the administrator “by chance”, but the framework would not be able to withstand intended and sophisticated attacks from a trusted application. As the source of a trusted application is usually known criminal behaviour could be prosecuted.

Each role must contain a definition of the default access right to devices (=top level resources) for which no permissions have been explicitly defined. It should also be specified if the owner of the role may install applications without further approval of the administrator. Usually this should be the case so that owners of a role can install updates of their applications.

The administrative interface must support:

- View/Create/delete of new roles (may be disabled by the provider of the framework)
- Assign/change signature to a role (this is necessary, when the owner of a role changes). In this case it should be possible to remove all applications installed by the old owner of the role from the framework. This function may be limited by the provider of the framework, e.g. if the system is provided based on a contract giving the provider a certain role. Usually the owner of the DSO cannot be changed.
- Assign permissions to resources for roles
- View/Install/uninstall applications (the administrator may be able to do this independently of a signature, if he/she trusts the application; this function may be disabled by the provider of the framework)
- Approve applications from roles that need individual approval for each new application
- View/Load/delete resource types (can also be done by applications via the resource administration)
- View/Create/delete resources (can also be done by applications via the resource administration)
- Change priority of resource demands registered by applications
- View/Change preferences of communication systems
- Change administration password and create additional user/password pairs to access user web interface via the internet
- View/change all resource values and application preferences values (optional)

Furthermore, the framework must offer an interface accessible over the internet which allows owners a role to send new/updated signed applications that will be installed automatically or after approval of the administrator. It should also be possible to remove applications via a signed command that were installed by the respective role. As the applications are signed encryption is not necessary, but might be of interest in some cases.
Other functions of the administrative interface cannot be accessed by applications. It may be possible, however, to provide configuration files that change a number of configurations without further interaction of the administrator. These files must be human readable and understandable for technically interested customers and they can be executed in the administration interface. These configuration files must also be signed when sent to the framework.

3. In-house Device Models

In the first line of each of the following tables, the purpose and meaning of each data model is described. In the second field of this line, the short name of the type is given that is used as type description when the type is used as an element of another data model. Basic types and other types listed in the specification of main components are not defined in this document.

Although the OSGi implementation of the framework is outside the scope of this document, the names and the units of the data elements are given here as these definitions might be used also by other implementations.

3.1. System Resource Types

3.1.1. Reference Types

The framework has to define references to various data types required by the services of the framework. These references might by names, IDs or references to objects depending on the implementation. The short name given in brackets will be used in the following models when part of a data model. The following reference types are needed:

- Reference to a resource type (ResTypeRef)
- Reference to a resource (ResourceRef)
- Reference to a communication system (ComSystemRef)
- Reference to a communication connection (ComConnectionRef)
- Reference to a time series set (TimeSeriesSetRef)
- Reference to a time series (TimeSeriesRef)
- Reference to a schedule definition (ScheduleDefRef)
- Reference to a continuous evaluation (EvalRef)
- Reference to an electric circuit of phase (ElCircuitRef)
- Reference to a room of the customer's premises (RoomRef)
- Basic Type (BaseType)

3.1.2. System Types

Schedule resource reference	SchedResRef		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name	String	yes	name	-

Time Series resource reference	TimeSeriesResRef		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of Time Series Set	String	yes	timeSeriesSet	-
Name of time series	String	yes	timeSeries	-

3.2. Common elements

Electric switch	ElSwitch		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
State	Boolean	yes	state	-

Controlled switch that does not switch electrical power on grid voltage level	NonElSwitch		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
State	Boolean	yes	state	-

Electric Multi-step-switch	ElMultSwitch		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Number of points on setpoint curve	Integer	yes	setPointNum	-
Power at each set point	Array of Analogue	yes	setPoint	W
Current power selected	Analogue	yes	state	W

Electric Step-less power control	ElControl		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Maximum power consumed	Analogue	yes	totWAvMaxConsume	W
Maximum power generated	Analogue	yes	totWAvMaxGen	W
Current power selected	Analogue	Yes	state	W

Electric Connection of device	ElConn		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Rated power	Analogue	yes	ratedPower	W
Measured power	Analogue	no	mmxPower	W
Circuit the device is connected to	ElCircuitRef	yes	elCircuitId	-

Electric Connection of device	ElConn		OSGI-impl.:	
Connection type: • fixed connection • plug, usually connected • plug, frequently disconnected (e.g. device connected via a manual switch) • plug, connected at various places (e.g. vacuum cleaner)	Enumerated	yes	connType	-

Temperature Storage	TempStor		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Rated operational temperature	Analogue	yes	stdOpTemp	K
Rated maximum temperature	Analogue	yes	maxstorTemp	K
Rated mimimum temperature	Analogue	yes	minstorTemp	K
Measured temperature	Analogue	no	mmxTemp	K

Outer physical dimensions and situation	PhysDim		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Height	Analogue	yes	height	m
Width	Analogue	yes	width	m
Depth	Analogue	yes	depth	m
Room where device is situated (negative number indicates a device installed outside, not in a room with walls and a roof)	RoomRef	yes	roomNumber	-
Integration type: • separate device, location may change regularly • separate device, but usually fixed location • integrated into in-house construction	Enumerated	yes	integrationType	-

Program Selection Unit	ProgUnit		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Number of programs available (length of arrays)	Integer	yes	progNum	-
Duration of each program	Time	yes	progDuration	-

Program Selection Unit	ProgUnit		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of each program	String	yes	progName	-
Choice of program (according to index of array)	Integer	yes	state	-

3.3. White Goods

Fridge	Fridge		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Cooling space parameters incl. sensor	TempStor	yes	coolSpace	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Cooling space size	Analogue	yes	ratedVolume	m3
Physical dimensions and situation	PhysDim	yes	dim	-

Freezer	Freezer		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Freezer space parameters incl. sensor	TempStor	yes	coolFreeze	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Freezing space size	Analogue	yes	ratedVolume	m3
Physical dimensions and situation	PhysDim	yes	dim	-

Cooling/Freezing Combination	FreezeCombi		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Cooling space parameters incl. sensor	TempStor	yes	coolSpace	-
Freezer space parameters incl. sensor	TempStor	yes	freezerSpace	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Cooling space size	Analogue	yes	ratedVolumeCool	m3
Freezing space size	Analogue	yes	ratedVolumeFreeze	m3
Physical dimensions and situation	PhysDim	yes	dim	-

Washing Machine	WashingMachine		OSGI-impl.:	
Name/Description	Type	Persistent	Name	unit
Available programs and selection of program	ProgUnit	yes	progUnit	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Capacity of clothes	Analogue	yes	laundryRating	kg
Physical dimensions and situation	PhysDim	yes	dim	-

Laundry Dryer	Dryer		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Available programs and selection of program	ProgUnit	yes	progUnit	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Capacity of clothes	Analogue	yes	laundryRating	kg
Physical dimensions and situation	PhysDim	yes	dim	-

Dish Washer	DishWasher		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Available programs and selection of program	ProgUnit	yes	progUnit	-
Switch	ElSwitch	yes	elSwitch	-
Electrical connection and measurement	ElConn	yes	elConn	-
Capacity in standard dish sets	Analogue	yes	dishRating	-
Physical dimensions and situation	PhysDim	yes	dim	-

3.4. Central Data

Electricity Price for consumption also used for generation if no other price is specified	PriceElectricityCon		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule for consumption in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Electricity Price for generation from CHP (including biomass fired if no other price is specified)	PriceElectricityCHP		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Electricity Price for generation from biomass	PriceElectricityBiomass		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Electricity Price for generation from biomass fired CHP plants	PriceElectricityBioCHP		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Note: If price schedules for CHP and Biomass are available but no schedule for biomass fired CHP usage of the correct schedule shall be up to an agreement between the operator of the plant and the buyer of the electricity. In case no agreement is available the operator shall have the right to choose whether the generation system is declared as „CHP“ or as „Biomass“.

Electricity Price for generation from photovoltaics.	PriceElectricityPV		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Electricity Price for any generation/delivery to the grid not specified by other generation prices	PriceElectricityGen		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

Price of delivery of natural gas	PriceNatGas		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Name of price schedule in persistent data storage	SchedResRef	yes	schedule	currency/ kWh

3.5. Heating, Ventilation and Cooling Consumption Devices

Heat generating unit	HeatGenUnit		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Rated thermal power	Analogue	yes	totWAvHeat	W
Thermal efficiency (heat energy/input energy)	Analogue	yes	efficiency	-
Start up time from cold start of heat generating unit to 90% of rated power	Time	yes	startUpCold	-
Start up time from warm start (output power just reduced to zero) to 90% of rated power	Time	yes	startUpWarm	-

Heat generating unit optional operational parameters	HeatGenUnitOpt		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Minimum run time	Time	yes	minRunTime	-
Minimum cool down time	Time	yes	minStopTime	-
Optimal run time (the device should be running for at least this time if not an emergency or beginning of period of very high price)	Time	yes	optRunTime	-
Cost of additional start up of the device (e.g. because of additional use of fuel and additional maintenance required)	Analogue	yes	switchCost	currency

This model can be added as an additional type as not all heat generating units make use of such detailed parameters

Heat connection	HeatConn		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Rated maximum flow	Analogue	yes	ratedFlow	m3/sec
Rated maximum temperature	Analogue	yes	ratedTemp	K
Heat circuit connected to	Integer	yes	heatCircuit	-
Measured input temperature	Analogue	no	mmxTempIn	K
Measured output temperature	Analogue	no	mmxTempOut	K
Measured flow	Analogue	no	mmxFlow	m3/sec
Measured power	Analogue	no	mmxPow	W

Indication of the heat circuit allows to model the way e.g. a cogeneration device, a heat buffer and an auxiliary heat burner are interconnected.

Heat storage physical	HeatStorPhys		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Capacity	Analogue	yes	capacity	Wh
Rated operational temperature	Analogue	yes	stdOpTemp	K
Rated maximum temperature	Analogue	yes	maxstorTemp	K
Rated minimum temperature	Analogue	yes	minstorTemp	K
Measured temperature	Analogue	no	mmxTemp	K
Heat connection	HeatConn	yes	heatConn	-

Heat storages may also just provide their state of charge:

Temperature Storage SOC	TempStorSOC		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Minimum level buffer	Analogue	yes	minSOC	%
Maximum level buffer	Analogue	yes	maxSOC	%
Measured level buffer (state of charge – SOC)	Analogue	no	mmxSOC	%
Heat connection	HeatConn	yes	heatConn	-

Ventilation	Ventilation		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Switch	ElSwitch	yes	elSwitch	-
Electric connection	ElConn	yes	elConn	-

Heat burner	HeatBurner		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Heat generation parameters	HeatGenUnit	yes	heatGenUnit	-
Switch	NonElSwitch	yes	switch	-
Heat connection	HeatConn	yes	heatConn	-
Physical dimensions and situation	PhysDim	yes	dim	-

Heat pump	HeatPump		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Heat generation parameters	HeatGenUnit	yes	heatGenUnit	-
Switch	ElSwitch	yes	elSwitch	-
Heat connection	HeatConn	yes	heatConn	-
Electric connection	ElConn	yes	elConn	-
Physical dimensions and situation	PhysDim	yes	dim	-

Electric Storage Heating	ElStorHeat		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Switch	ElSwitch	yes	elSwitch	-
Ventilation for transfer of heat from storage to room	Vent	yes	ventilation	-
Electric connection	ElConn	yes	elConn	-
Measured storage temperature	Analogue	yes	mmxTemp	-
Physical dimensions and situation	PhysDim	yes	dim	-

Air Conditioning	AirCond		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Switch	ElSwitch	yes	elSwitch	-
Electric connection	ElConn	yes	elConn	-
Physical dimensions and situation	PhysDim	yes	dim	-

Temperature Sensor	TempSens		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Measured temperature	Analogue	no	mmxTemp	-
Physical dimensions and situation (provides information where installed, room number or outside sensor)	PhysDim	yes	dim	-

3.6. Generation Devices

Cogeneration Device	CHP		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Heat generation parameters	HeatGenUnit	yes	heatGenUnit	-
Switch	ElSwitch	yes	elSwitch	-
Heat connection	HeatConn	yes	heatConn	-
Electric connection	ElConn	yes	elConn	-
Physical dimensions and situation	PhysDim	yes	dim	-
Electrical start up time from cold start to 90% of rated electrical power	Time	yes	elStartUpCold	-
Electrical start up time from warm start (output power just reduced to zero) to 90% of rated electrical power	Time	yes	elStartUpWarm	-
Electric consumption at start-up (total electrical energy consumption from grid during start-up process)	Analogue	yes	startElEnergy	Wh
Electric consumption at start-up (peak power consumed from grid during start-up process)	Analogue	yes	startElPower	W
Fuel • natural gas • oil • solid biomass • vegetable oil • biogas	Enumerated	yes	fuel	-

Electrical efficiency and overall efficiency can be calculated from rated electrical, thermal power and thermal efficiency.

PV plant	PVplant		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Switch	ElSwitch	yes	elSwitch	-
Electric connection	ElConn	yes	elConn	-
Power limit (requires derating if production would be higher based on current solar irradiation)	Analogue	no	powerLimit	W
Physical dimensions and situation	PhysDim	yes	dim	-

3.7. Metering Data

Electrical meter profile	ElMeter		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Metered profile	SchedResRef	yes	schedule	kWh
Electrical circuit metered	Integer	yes	elCircuitId	-

Natural Gas meter profile	GasMeter		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Metered profile	SchedResRef	yes	schedule	kWh
Name of device or device group metered	String	yes	deviceName	-

Heat meter profile	HeatMeter		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Metered profile	SchedResRef	yes	schedule	kWh
Heat circuit metered	Integer	yes	heatCircuit	-

Water meter profile	WaterMeter		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Metered profile	SchedResRef	yes	schedule	kWh
Description of water circuit metered	String	yes	circuitName	-

3.8. Forecast Data

Outside temperature forecast	TempForecast		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Forecast outside temperature profile at the location of the premises	SchedResRef	yes	schedule	K

Solar irradiation forecast	SolarForecast		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Forecast of solar irradiation at the location of the premises	SchedResRef	yes	schedule	W
Angle of inclination of a module directing to south the forecast is made for	Analogue	yes	schedule	°

Wind speed forecast	WindForecast		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Wind speed forecast at the location of the premises	SchedResRef	yes	schedule	m/s
Altitude above ground the forecast is made for	Analogue	yes	schedule	m

Humidity forecast	HumidityForecast		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Humidity of outside air at the location of the premises	SchedResRef	yes	schedule	%

Precipitation forecast	PrecipitationForecast		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
Precipitation forecast at the location of the premises	SchedResRef	yes	schedule	mm

3.9. Energy Management Parameters

Fixed Program Shift Limits	FPSLimit		OSGI-impl.:	
Name/Description	Type	Persistent	name	unit
First start time allowed	Time	yes	startTime	-
Final time the program has to be finished latest	Time	yes	endTime	-
Maximum program shift: As an alternative to the definition of a time span of operation also a maximum shifting (or just separation of the device from power) can be defined.	Time	yes	maxShiftTim e	-
Management mode: • operate within time span defined by startTime and endTime • operate by maximum program shift	Enumerated	yes	endTime	-
Priority (in case of emergency programs may not be processed in the time span defined before, but the programs will be processed based on priority levels, priority level 0 will be processed first)	Integer	yes	priority	-
Maximum price: In case the program cannot be processed in the time span specified before for the maximum price or lower, it shall be shifted to another time.	Analogue	yes	endTime	currency

4. Development of an Open Standard and Reference Implementation for In-House Services

In order to define and develop a standard for the in-house services described above, Fraunhofer IWES has started the Open Gateway for Energy Management Alliance in September 2009. The scope of this Alliance is to provide an open software framework for energy management in the building sector, including private buildings and households. This framework is to be run on a central building gateway which serves as interface between the Smart House and the Smart Grid, integrating as many applications in the area of energy management and energy efficiency as possible. The development of a reference implementation for this framework is also goal of OGEMA and is currently ongoing as part of IWES SmartHouse/SmartGrid activity. The field test B within project SmartHouse/SmartGrid will in fact be the first test of the newly developed framework in real-field at customer's premises.

An overview of the frameworks internal organization has already been given in chapter 2.1. It has become apparent that a very important aspect of this framework is its openness both in terms of software development as well as manufacturer independency. For the software development, both the OGEMA specification as well as the reference implementation will be provided as open source and be made freely downloadable. In this way, developers who are not members of the alliance can also contribute on new applications or communication system drivers. This can be compared with the open source paradigm of, for example, the Firefox web browser or the Linux operating system. Software developers again will be supported by the possibility to use the framework's open interfaces. This allows, for example, for programming a new web interface for device management without deep knowledge of the hardware driver design for device control. Again, this is similar to the functionality of an operating system which offers specified interfaces for device access to the application layer.

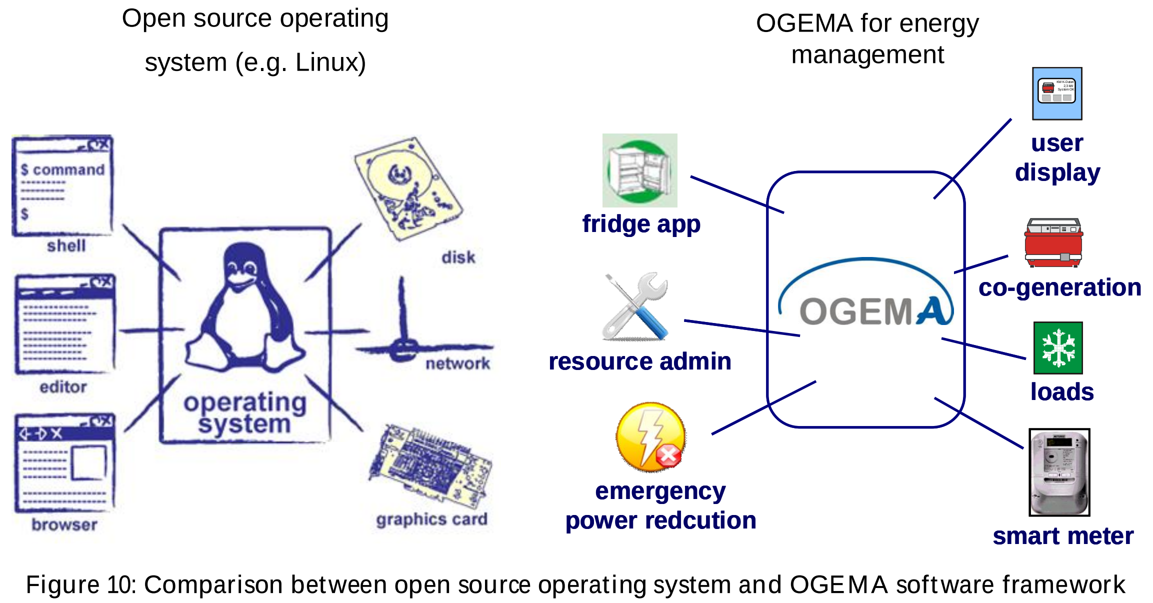


Figure 10: Comparison between open source operating system and OGEMA software framework

The open-source and open specification approach also ensures manufacturer independency, since different applications from various manufacturers will be able to run on the framework as long as each application is complying with the specification interfaces. Therefore, even manufacturers who are not alliance members can program applications. This shall allow a multitude of applications („apps") to be developed within a short period of time. These apps shall cover the differing requirements from private households, super markets, small businesses up to public institutions such as schools and hospitals and help to tap potentials for energy efficiency not accessed today. However, at least in the starting phase of the alliance it will be recommended to application designers and manufacturers to become alliance members. This way, they will be able to influence the further development of the specification and take part in discussions on new specification requirements, to receive early information on specification draft documents, to access support from an OGEMA contact office, to list own products on the alliance's web site and to benefit from moderation by Fraunhofer IWES for the advance of the OGEMA specification with the goal of a maximum wide usage for a sustainable energy supply while offering a maximum of neutrality.

The OGEMA framework's main features are summarized as follows:

- OGEMA uses well-known widely-accepted software standards for its execution environment that are available as open source and as commercial products. For example, the current reference implementation is based on Java and OSGi.
- OGEMA allows for applications to be developed not knowing how devices are connected to the system and communication drivers can be developed independently from the application that uses the driver. Data Models defined by the OGEMA specification act as an intermediary layer allowing both applications and drivers to be developed against an interface definition that is independent from the actual hardware communication connections and from the actual applications deployed.
- Representation of devices and common services by extensible data models. The OGEMA framework provides a fixed number of services needed to register/unregister device types, actual devices, applications and communication drivers. Also services for the Plug&Play-functionality, for application runtime control, for logging and for getting information on components registered, methods for reading and writing device data are included. New device types can be installed dynamically by providing appropriate Java classes containing new data structures to the framework.
- Open interface for software applications and hardware/communication drivers. The entire API of the OGEMA framework including interfaces for software applications and communication drivers as well as sample code will be made public on the OGEMA web site.
- Support for various in-house communication systems. By defining a common interface layer for ICT used in-house, the "communication system", OGEMA provides a basis for including market-available technologies such as ZigBee, Z-Wave or KNX into the energy management system. This also allows OGEMA applications to integrate or interact with building automation systems.
- Plug&Play support: the OGEMA framework provides methods for minimizing installation effort for the inclusion of new devices into the energy management. The OGEMA interface for communication drivers allows for various communication concepts and levels of auto-detection of devices. This ranges from the completely automatic device discovery to user interaction for providing the needed installation information to the framework (e.g. the information that a newly connected temperature sensor is situated in the bathroom, where temperature is configured to be highest in the mornings and evenings).
- "Firewall" between public grid and customer grid: the gateway acts as a firewall between the public and the private communication systems allowing only the interaction between the systems as defined by the gateway configuration.
- Resource control based on user-specific access rights and permissions. OGEMA user accounts on the gateway grant specific device access and applications permissions.
- Web-based Man-Machine Interface support. OGEMA uses standard web technology to implement the user interface. So the user can access the interface by any web-enabled device with an internet browser. Applications can bring their own web pages using HTML and JSP (Java Server Pages for dynamic page content).

The first version of the specification and reference implementation shall be made ready for download at the OGEMA webpage www.ogema-alliance.org in early 2010. Also, updated information about the alliance's progress will be published there. A technical documentation of the reference implementation is also part of the SmartHouse/SmartGrid deliverable D2.3.

Annex: Detailed Description of In-House Services

1. Services of the Resource Administration

1.1. Management of Resource Types

Aspect	Specification
Service name	addResourceType
Location of service	ResourceAdministrationService
Service functionality	Add Resource Type to the framework
Service input / output	Input: Resource Type Description Output: (Resource Type Reference if not the same as resource type name)
Service users	- Administration interface - Applications that require specific type(s)
Expected frequency of use	Frequently during setup of framework otherwise: < 1x / day
Related use cases	all

Aspect	Specification
Service name	deleteResourceType
Location of service	ResourceAdministrationService
Service functionality	Delete Resource Type from the framework
Service input / output	Input: Resource Type Reference Output: -
Service users	- Administration interface - Applications that require specific type(s)
Expected frequency of use	infrequently
Related use cases	all (upon uninstall)

Aspect	Specification
Service name	getResourceTypesInstalled
Location of service	ResourceAdministrationService
Service functionality	Get all resource types available on the framework
Service input / output	Input: - Output: List of all resource types (array of String)

Service users	- Administration interface - Applications that require specific type(s)
Expected frequency of use	infrequently
Related use cases	Administration Using manufacturer specific functionality

Aspect	Specification
Service name	getTypeDescription
Location of service	ResourceAdministrationService
Service functionality	Get description of Resource Type
Service input / output	Input: Resource Type Reference Output: Resource Type description
Service users	- Administration interface - Applications that require specific type(s)
Expected frequency of use	infrequently
Related use cases	Administration Using manufacturer specific functionality

1.2. *Management of Resources*

Aspect	Specification
Service name	addResource
Location of service	ResourceAdministrationService
Service functionality	Add Resource representation
Service input / output	Input: Resource Type Reference, name of new top-level resource Output: Resource Reference
Service users	- Administration interface - Communications Systems importing new resources - Applications installing new devices
Expected frequency of use	Frequently during setup of framework otherwise: < 1x / day
Related use cases	all

Aspect	Specification
Service name	deleteResource
Location of service	ResourceAdministrationService

Service functionality	Delete Resource representation from the framework
Service input / output	Input: Resource Reference Output: -
Service users	- Administration interface - Communication Systems detecting devices that are removed
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	addResourceAdditionalType
Location of service	ResourceAdministrationService
Service functionality	Add an optional sub-resource to an existing resource
Service input / output	Input: Parent of new sub-resource (Resource Reference), Resource Type Reference (must refer to an “Additional Resource Type”) Output: Resource Reference
Service users	- Administration interface - Applications installing new devices)
Expected frequency of use	infrequently
Related use cases	Using manufacturer specific functionality

Aspect	Specification
Service name	getResourceName
Location of service	ResourceAdministrationService
Service functionality	Get name of Resource based on its reference
Service input / output	Input: Resource reference Output: Name (String)
Service users	- Administration interface - User interface
Expected frequency of use	Depending on user interface usage
Related use cases	all

Aspect	Specification
Service name	getResourceType
Location of service	ResourceAdministrationService

Service functionality	Get type of a resource
Service input / output	Input: Resource reference Output: Resource Type reference
Service users	- Administration interface - User interface - Applications searching for resource types using wildcards
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getSuperResourceId
Location of service	ResourceAdministrationService
Service functionality	Get parent of a resource
Service input / output	Input: Resource Type Reference Output: Parent of Input (Resource Type Reference), error if input was top-level resource
Service users	- Administration interface - Applications that can use various resource types based on a specific sub-resource element
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getAttributeId
Location of service	ResourceAdministrationService
Service functionality	Get reference to a sub-resource based on its name, which is defined by the Resource Type description
Service input / output	Input: Resource Reference, name of sub-resource (String) Output: Resource Reference of sub-resource
Service users	- Administration interface - Applications
Expected frequency of use	Frequently during setup of framework otherwise: < 1x / day
Related use cases	all

Aspect	Specification
Service name	getToplevelResourceId
Location of service	ResourceAdministrationService
Service functionality	Get device or other top-level resource by its name or by a Reference of a sub-resource
Service input / output	Input: Device name (String) or resource Reference Output: Resource Reference of top-level resource
Service users	- User Interface - Applications that can use various resource types based on a specific sub-resource element (e.g. a temperature storage that might be found in various devices)
Expected frequency of use	infrequently
Related use cases	all

1.3. *Search for Resources*

Aspect	Specification
Service name	getResourcesIds
Location of service	ResourceAdministrationService
Service functionality	Get all resources that fit one of the Resource Types given (should also support wild cards like '*' and '?')
Service input / output	Input: List of Resource Type names Output: List of Resource References
Service users	- Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getChildResourcesIds
Location of service	ResourceAdministrationService
Service functionality	Get all sub-resources of a resource
Service input / output	Input: Resource Type Reference Output: Sub-resources of input (List of Resource Type References), empty if input is a basic type resource
Service users	- Administration interface - Applications using optional sub-resources
Expected frequency of use	infrequently
Related use cases	Administration Using manufacturer specific functionality

1.4. *Read / Write of Resources*

Aspect	Specification
Service name	setResourceElementBoolean
Location of service	ResourceAdministrationService
Service functionality	Write into resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Boolean), Value to write Output: -
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	setResourceElementInteger
Location of service	ResourceAdministrationService
Service functionality	Write into resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Integer), Value to write Output: -
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	setResourceElementAnalogue
Location of service	ResourceAdministrationService
Service functionality	Write into resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Analogue), Value to write Output: -
Service users	- User interface - Applications - Communication Systems

Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	setResourceElementString
Location of service	ResourceAdministrationService
Service functionality	Write into resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type String), Value to write Output: -
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	setResourceElementTime
Location of service	ResourceAdministrationService
Service functionality	Write into resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Time), Value to write Output: -
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	getResourceElementBoolean
Location of service	ResourceAdministrationService
Service functionality	Read from resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Boolean) Output: Value of resource
Service users	- User interface

	- Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	getResourceElementInteger
Location of service	ResourceAdministrationService
Service functionality	Read from resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Integer) Output: Value of resource
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	getResourceElementAnalogue
Location of service	ResourceAdministrationService
Service functionality	Read from resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Analogue) Output: Value of resource
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	getResourceElementString
Location of service	ResourceAdministrationService
Service functionality	Read from resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type String)

	Output: Value of resource
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

Aspect	Specification
Service name	getResourceElementTime
Location of service	ResourceAdministrationService
Service functionality	Read from resource of basic type
Service input / output	Input: Resource Type Reference (must refer to resource of simple type Time) Output: Value of resource
Service users	- User interface - Applications - Communication Systems
Expected frequency of use	Frequently
Related use cases	all

1.5. *Management of Communication Systems*

Aspect	Specification
Service name	registerCommunicationSystem
Location of service	ResourceAdministrationService
Service functionality	Register communication system that has been installed on the system
Service input / output	Input: Communication System Reference Output: -
Service users	- Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	unregisterCommunicationSystem
Location of service	ResourceAdministrationService
Service functionality	Release communication system

Service input / output	Input: Communication System Reference Output: -
Service users	- Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getCommunicationSystems
Location of service	ResourceAdministrationService
Service functionality	Get a list of all communication systems registered
Service input / output	Input: - Output: List of references to communication systems
Service users	- Administration interface - Applications that perform binding of resources to communication connections
Expected frequency of use	infrequently
Related use cases	all

1.6. Management of Resource Demand of Applications

Aspect	Specification
Service name	registerResourceDemand
Location of service	ResourceAdministrationService
Service functionality	Register resource demand (see Main Component Specification)
Service input / output	Input: Description of resource demand (by type as in getResources, by resource name or by resource reference), resource demand reference, connection type (notification on status only, read, read/write), priority, name of resource demand, Callback Reference for notification informed of new resources fitting a demand, changes of the demand fulfilment and changes of the values of the resources that are assigned to an application Output: -
Service users	- Applications
Expected frequency of use	Frequently during setup of applications otherwise: < 1x / day
Related use cases	all

Aspect	Specification
Service name	registerSecondaryDemand
Location of service	ResourceAdministrationService
Service functionality	Register demand for an additional resource that is required for using a primary resource the application get access because it fits a demand. This can be used e.g. when access to a switch is required together with access to a sensor. A secondary demand must always point to a dedicated resource, no to a type. The secondary resource will be released as soon as the primary resource is released. If the secondary resource loses the lock also the primary resource will be released.
Service input / output	Input: Resource Demand Description as in register resource demand, reference to primary resource demand Output: -
Service users	- Applications
Expected frequency of use	Frequently during setup of framework otherwise: < 1x / day
Related use cases	all

2. Persistent Storage

2.1. *Preferences*

Aspect	Specification
Service name	createPreferencesSet
Location of service	Preferences
Service functionality	Open set of preferences/parameters for an application. Create the set if it does not exist yet.
Service input / output	Input: Preferences Set Name Output: Preferences Set Reference
Service users	- Applications
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	deletePreferencesSet
Location of service	Preferences

Service functionality	Delete set of preferences/parameters for an application
Service input / output	Input: Preferences Set Name Output: -
Service users	- Applications
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getInt
Location of service	Preferences
Service functionality	Read Integer from set of preferences, return standard value if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (Integer) Output: Value read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getIntArray
Location of service	Preferences
Service functionality	Read array of Integers from set of preferences, return array with standard length consisting only of standard values if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (Integer), length of standard array Output: Array read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getFloat
Location of service	Preferences
Service functionality	Read Analogue from set of preferences, return standard value if it does not exist

Service input / output	Input: Preferences Set Reference, preference value name, standard value (Float) Output: Value read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getFloatArray
Location of service	Preferences
Service functionality	Read array of Floats from set of preferences, return array with standard length consisting only of standard values if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (Float), length of standard array (Integer) Output: Array read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getString
Location of service	Preferences
Service functionality	Read String from set of preferences, return standard value if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (String) Output: Value read
Service users	- Applications
Expected frequency of use	Frequently
Related use cases	All

Aspect	Specification
Service name	getStringArray
Location of service	Preferences
Service functionality	Read array of Strings from set of preferences, return array with standard length consisting only of standard values if it does not exist

Service input / output	Input: Preferences Set Reference, preference value name, standard value (String), length of standard array (Integer) Output: Array read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getTime
Location of service	Preferences
Service functionality	Read Time-value from set of preferences, return standard value if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (Time) Output: Value read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	getTimeArray
Location of service	Preferences
Service functionality	Read array of Time values from set of preferences, return array with standard length consisting only of standard values if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name, standard value (Time), length of standard array (Integer) Output: Array read
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	setInt
Location of service	Preferences
Service functionality	Write Integer into set of preferences, create value if it does not exist yet

Service input / output	Input: Preferences Set Reference, preference value name Output: Value to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	All

Aspect	Specification
Service name	setIntArray
Location of service	Preferences
Service functionality	Write array of Integers into set of preferences, create it if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name Output: Array to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	All

Aspect	Specification
Service name	setFloat
Location of service	Preferences
Service functionality	Write Analogue into set of preferences, create value if it does not exist yet
Service input / output	Input: Preferences Set Reference, preference value name Output: Value to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	All

Aspect	Specification
Service name	setFloatArray
Location of service	Preferences
Service functionality	Write array of Analogues into set of preferences, create it if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name Output: Array to write
Service users	- Applications
Expected frequency of use	frequently

Related use cases	all

Aspect	Specification
Service name	setString
Location of service	Preferences
Service functionality	Write String into set of preferences, create value if it does not exist yet
Service input / output	Input: Preferences Set Reference, preference value name Output: Value to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	setStringArray
Location of service	Preferences
Service functionality	Write array of Strings into set of preferences, create it if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name Output: Array to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	setTime
Location of service	Preferences
Service functionality	Write Time value into set of preferences, create value if it does not exist yet
Service input / output	Input: Preferences Set Reference, preference value name Output: Value to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification

Service name	setTimeArray
Location of service	Preferences
Service functionality	Write array of Time values into set of preferences, create it if it does not exist
Service input / output	Input: Preferences Set Reference, preference value name Output: Array to write
Service users	- Applications
Expected frequency of use	frequently
Related use cases	all

2.2. *Schedules*

Aspect	Specification
Service name	createScheduleSeries
Location of service	dbScheduleService
Service functionality	Create schedule series in persistent storage
Service input / output	Input: Schedule Series Definition Output: Schedule Series Reference (if not the same as name)
Service users	- Administration interface - Applications using own schedules - Communication Systems importing schedules
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	deleteScheduleSeries
Location of service	dbScheduleService
Service functionality	Delete schedule series from persistent storage
Service input / output	Input: Schedule Series Reference Output: -
Service users	- Administration interface
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	getScheduleSeriesDefintion
Location of service	dbScheduleService
Service functionality	Check whether Schedule Series Reference exists and read it
Service input / output	Input: Schedule Series Reference Output: Schedule Series Defintion
Service users	- Administration interface
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	addScheduleFixedStep
Location of service	dbScheduleService
Service functionality	Add new schedule information into a Schedule Series; overwrites existing schedule information for the same period of time
Service input / output	Input: Single Schedule (including information of Schedule Series Reference) Output: -
Service users	- Applications - Communication Systems
Expected frequency of use	a few times per day (may be much more frequently in some cases, e.g. emergencies)
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	getScheduleFixedStep
Location of service	dbScheduleService
Service functionality	Get schedule information for scheduling interval containing start time
Service input / output	Input: Schedule Series Reference, start time of scheduling period (Time) Output: Single Schedule
Service users	- Applications - Communication Systems
Expected frequency of use	a few times per day (may be much more frequently in

	some cases, e.g. emergencies)
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	getValueFromSchedule
Location of service	dbScheduleService
Service functionality	Get single value from all schedule information in a Schedule Series for a certain time
Service input / output	Input: Schedule Series Reference, time Output: value (Analogue)
Service users	- Administration interface - Applications using own schedules - Communication Systems importing schedules
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

2.3. *Time Series*

Aspect	Specification
Service name	openTimeSeriesSet
Location of service	dbTimeSeriesService
Service functionality	Open or create Time Series Set. In case the time series set already exists, the existing time series shall be opened using the new configuration otherwise a new time series set shall be created. Each time series can only be opened by a single application at a time.
Service input / output	Input: Time Series Set name, Time Series Set Definition Output: Time Series Set Reference
Service users	- Logging and evaluation
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	closeTimeSeriesSet
Location of service	dbTimeSeriesService

Aspect	Specification
Service functionality	Close Time Series Set. The Time Series Set is then available to be opened by another application.
Service input / output	Input: Time Series Set name Output: -
Service users	- Logging and evaluation
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	getTimeSeriesSetDefinition
Location of service	dbTimeSeriesService
Service functionality	Get Time Series Set Definitoin
Service input / output	Input: Time Series Set name Output: Time Series Set Definition
Service users	- Logging and evaluation
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	deleteTimeSeriesSet
Location of service	dbTimeSeriesService
Service functionality	Delete Time Series Set
Service input / output	Input: Time Series Set name Output: Time -
Service users	- Logging and evaluation
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	addTimeSeriesData
Location of service	dbTimeSeriesService
Service functionality	Add one set of data consisting of one time stamp and one

Aspect	Specification
	value for each time series. The time stamp must be later then the time stamp written previously, so no data that was written before can be changed using this service (use writeTimeSeriesValue... instead). In case the TimeSeriesSet specifies a reduction mode, the reduction will be processed by this service.
Service input / output	Input: Time Series Set Reference, Time stamp, new values (array of Analogues) Output: -
Service users	- Logging and evaluation
Expected frequency of use	frequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	writeTimeSeriesValue
Location of service	dbTimeSeriesService
Service functionality	Write into existing data of a time series. It is not possible to add any time stamps by this function, but existing data is overwritten. Usually the closest time stamp available should be used. If the time is outside the range of the time series, however, an error should be returned.
Service input / output	Input: Time Series Set Reference, Time stamp, name of time series, new value (Analogue) Output: -
Service users	- Logging and evaluation
Expected frequency of use	frequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	readTimeSeriesRange
Location of service	dbTimeSeriesService
Service functionality	Read all data series from one time series set in a certain time range based on the name of the time series (reading unsynchronized with writing)
Service input / output	Input: Time Series Set name, start time, end time, names of time series to read (array of Strings), maximum number of values to read per time series (Integer) Output: Map of time series names and Arrays of the

Aspect	Specification
	values in the range. Also an array of time stamps is returned.
Service users	- Logging and evaluation
Expected frequency of use	frequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	readTimeSeriesRangeAligned
Location of service	dbTimeSeriesService
Service functionality	Read in a certain time range from several time series being part of one or more time series sets and align the data.
Service input / output	Input: Array of Time Series Set Requests (consisting of the input variables as defined in readTimeSeriesRange except for start and end time), start time, end time, mode of alignment (fixed step, as many time stamps as make sense, use first time series set to define common time stamps) (Enumerated), in case of fixed step: step size (Time) Output: Map of time series names and Arrays of the values in the range. Also an array of time stamps is returned.
Service users	- Logging and evaluation
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	commitToDiskConfig
Location of service	dbTimeSeriesService
Service functionality	Configure how often data shall be written to persistant storage (in case system supports buffering of data in RAM) or initiate immediate writing
Service input / output	Input: Only write to persistent storage when commitNewOperationsToDisk(true) is called / perform regular writing to persistent storage every <interval> seconds (if any operations pending) (Enumerated), time between two automated writings to persistent storage (0: do not allow buffering) (Time)

	Output: -
Service users	- Administration interface - Applications
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

Aspect	Specification
Service name	commitNewOperationsToDisk
Location of service	dbTimeSeriesService
Service functionality	Enable/disable RAM buffering of data for persistent storage
Service input / output	Input: Boolean (true: write all data buffered in RAM into persistent storage and do not allow for any more buffering until service is called again to switch to false). While buffering is disabled the configuration set by commitToDiskConfig are not relevant. Output: number of write operations currently buffered (Integer)
Service users	- Administration interface - Applications
Expected frequency of use	infrequently
Related use cases	e.g. management using price / device operation schedules

3. Communication System

In case a resource represents not a hardware device, the term "hardware" in the following service descriptions stands for the remote data base, e.g. for the system providing electricity prices.

Aspect	Specification
Service name	getName
Location of service	CommunicationSystem
Service functionality	Get name of communication system
Service input / output	Input: - Output: Name (String)
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	All

Aspect	Specification
Service name	getMinimalTimeStep
Location of service	CommunicationSystem
Service functionality	Get minimal time between readings of a potential new sensor
Service input / output	Input: - Output: Minimal time step (Time)
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	isMinimalTimeStepVariable
Location of service	CommunicationSystem
Service functionality	Determine whether the result of getMinimalTimeStep may change e.g. when additional sensors are added or when the quality of a radio connection changes
Service input / output	Input: - Output: Boolean
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	setStandardTimeStep
Location of service	CommunicationSystem
Service functionality	Set standard time step that will be used for sensors connected by the communication system
Service input / output	Input: Time Output: -
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getStandardTimeStep
Location of service	CommunicationSystem
Service functionality	Get standard time step
Service input / output	Input: - Output: Time
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	getConfigurationData
Location of service	CommunicationSystem
Service functionality	Get reference to internal configuration data of communication system. In general this will be a data structure specific to the communication system.
Service input / output	Input: - Output: Reference to internal configuration data of communication system
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	configurationUpdated
Location of service	CommunicationSystem
Service functionality	The service should be called when the configuration data (previously provided as reference by getConfigurationData) has been changed.
Service input / output	Input: - Output: -
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	checkAddress
Location of service	CommunicationSystem
Service functionality	Check if address is valid for communication system
Service input / output	Input: Address (String) Output: Enumerated (address ok; format valid, but address not reachable; format invalid)
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	connectResourceToAddress
Location of service	CommunicationSystem
Service functionality	Link an internal resource to a remote value that is identified by an address. This can be used if the ComSystem has not identified the value to be accessible e.g. because the ComSystem has no auto-detect functionality
Service input / output	Input: - Address (String) - Reference to resource to connect to - pollType (Enumerated): - 1: The ComSystem is responsible for writing into the resource according to the cycle specified or must check itself; whether a value to send to the hardware has been changed for reading - 2: The service updateConnValue is called each time a communication to the hardware shall take place - 3: server mode / no polling - Enumerated(hardware to resource (read from hardware); resource to hardware (write to hardware); both directions) - Time between update of value (Time) (only relevant for pollType=1; if zero use standard step time) Output: ComConnectionRef
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all

Aspect	Specification
Service name	updateConnValue
Location of service	CommunicationSystem
Service functionality	Notify the communication system that it needs to update a resource or send some new value to the hardware
Service input / output	Input: ComConnectionRef Output: -
Service users	- Resource administration
Expected frequency of use	frequently
Related use cases	all

Aspect	Specification
Service name	UnbindResource
Location of service	CommunicationSystem
Service functionality	Close link of external and internal data
Service input / output	Input: ComConnectionRef Output: -
Service users	- User interface - Administration interface
Expected frequency of use	infrequently
Related use cases	all